

La arquitectura propuesta se basa en un modelo de distribución de cómputo por delegación, diseñado para resolver la ecuación cuadrática mediante la segmentación de tareas en nodos especializados. En este ecosistema, el Cliente actúa como el iniciador del flujo, proporcionando las constantes a , b y c , pero manteniendo un aislamiento total de la lógica de cálculo, comunicándose exclusivamente con un Servidor de Operaciones Central.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

El Servidor de Operaciones finge como el núcleo orquestador y centro de control del sistema. Su responsabilidad no es el cálculo matemático, sino la gestión del estado de la transacción y la coordinación del flujo de datos. Este nodo dirige las peticiones de forma secuencial hacia tres nodos esclavos: el Servidor 1 (Resuelve la Raíz del nominador), el Servidor 2 (Adición/Sustracción) y el Servidor 3 (Denominador y División Final). La integridad de la arquitectura reside en que los nodos de operación son atómicos y no poseen comunicación entre sí, garantizando que el orquestador sea el único responsable de la sincronización y el manejo de los resultados intermedios.

Finalmente, el flujo operativo sigue una lógica de tubería distribuida (pipeline), donde el orquestador recibe los parámetros, invoca al Servidor 1 para obtener el discriminante, traslada ese resultado al Servidor 2 para las variantes de la suma/resta, y culmina con el Servidor 3 para el cálculo del denominador y la división final. Una vez consolidada la información, el Servidor de Operaciones cierra el ciclo retornando las dos raíces finales al cliente a través de sockets. Lo anterior, se representa en los siguientes diagramas:

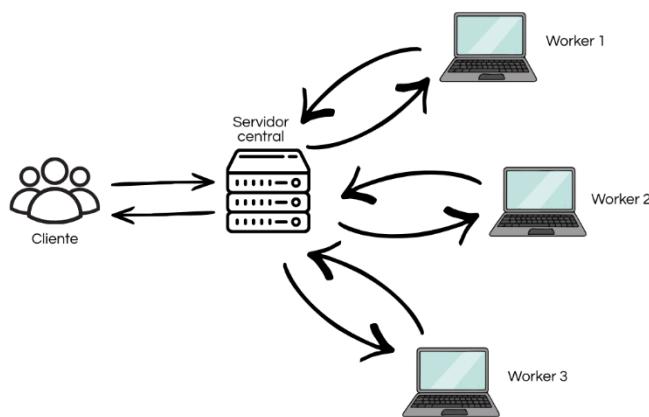


Figura 1. Modelo arquitectónico

Caso sin fallos:

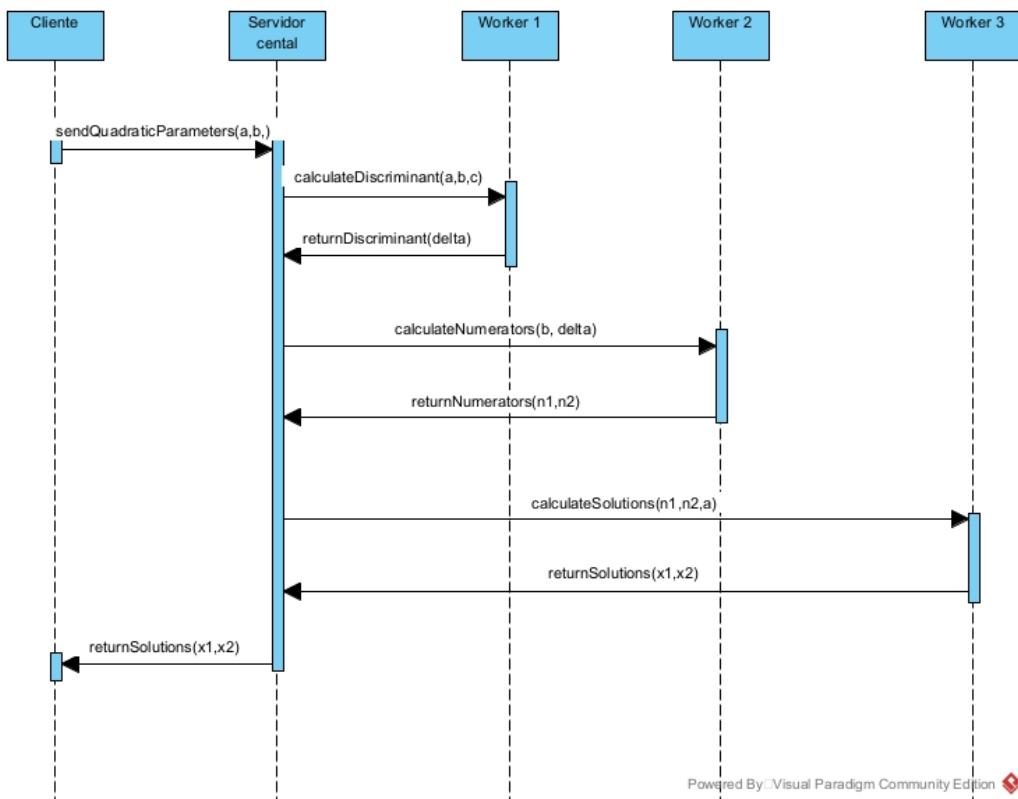


Figura 2. Diagrama de secuencia normal

Flujo Principal

El cliente envía a, b, c al coordinador vía socket TCP (función main() del cliente, serializado como JSON). El coordinador llama a process_quadratic_request, que ejecuta tres etapas en secuencia usando call_with_failover:

1. Discriminante → envía op = "sqrt_discriminant" a op1. Devuelve sqrt_d.
2. Numeradores → envía op = "numerator" con b y sqrt_d a op2. Devuelve num_plus, num_minus.
3. División → envía op = "division" con los numeradores y a a op3. Devuelve x1, x2.
4. Ningún worker falla, call_with_failover retorna al primer intento en cada etapa. El coordinador responde al cliente con x1, x2 y modo “pipeline”.

Flujos alternos en validación (antes de contactar cualquier worker):

- Parámetros no numéricos: si alguno de los valores no puede convertirse a número, el coordinador retorna inmediatamente un JSON de error al cliente sin contactar ningún worker.
- a = 0: si el coeficiente cuadrático es cero, el coordinador retorna un error indicando que la ecuación no es cuadrática.

Si la validación pasa, se ejecutan las tres etapas:

1. Discriminante → envía op = "sqrt_discriminant" a op1. Devuelve sqrt_d.

- a. Flujo alterno — discriminante negativo: si $b^2 - 4ac < 0$, Worker 1 devuelve las raíces como números complejos (parte real e imaginaria separadas). El coordinador los propaga sin modificación hasta el cliente.
2. Numeradores → envía op = "numerator" con b y sqrt_d a op2. Devuelve num_plus, num_minus.
3. División → envía op = "division" con los numeradores y a a op3. Devuelve x1, x2.

El coordinador responde al cliente con x1, x2 y modo "pipeline".

Flujo con fallback (tolerancia a fallos):

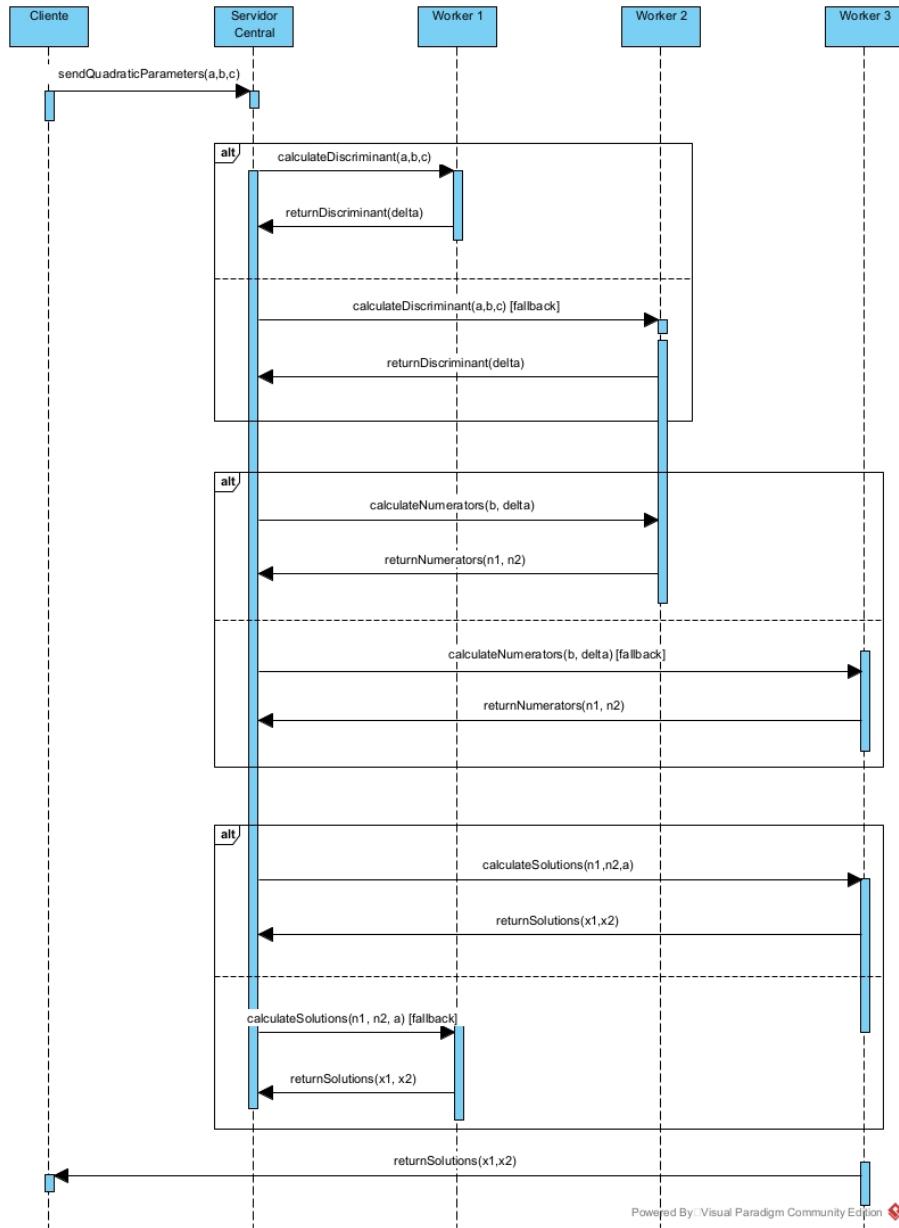


Figura 2. Diagrama de secuencia con fallback

Flujo Principal

El cliente envía igual que en el caso sin fallos, el coordinador ejecuta las mismas tres etapas, pero call_with_failover maneja los fallos así:

- Si un worker no responde (timeout o error de conexión), se añade a `dead_ops` y se pasa al siguiente candidato en el orden del `ROLE_PLAN`.

- Esto se repite por cada etapa hasta encontrar un worker vivo.

Si en cualquier momento `len(dead_ops) >= 2`, el coordinador abandona el pipeline y llama a `call_full_quadratic_on_single_alive`, que envía `op = "full_quadratic"` al único worker superviviente para que realice todo el cálculo internamente.

El coordinador responde al cliente con `x1, x2, la lista de workers en dead_ops, y modo "pipeline" o "single_nodeFallback" según corresponda.`

Comunicación y formato de los mensajes

Para establecer la comunicación entre componentes del sistema se usa sockets TPC, los cuales garantizan que una conexión estable y que los datos lleguen en orden, como formato de intercambio de datos se usó JSON debido a que es legible y fácil de manejar en diferentes lenguajes.

Cada mensaje es un objeto JSON que se envía como una cadena de texto terminada con el carácter de nueva línea (\n) al final que hace de separador, esto permite que el receptor saber exactamente donde termina un mensaje y donde inicia el siguiente, para no repetir este proceso se utilizan las funciones auxiliares `send_json()` y `recv_json` donde la primera convierte el objeto a JSON, le añade el \n y lo envía a `recv_json()`, que va acumulando bytes hasta encontrar el delimitador, decodifica la línea como JSON y retorna el objeto resultante.

Ahora bien, como los servidores encargados de operaciones no se comunican entre ellos, para garantizar la robustez del sistema frente a fallos de red o caídas de nodos, el Servidor de Operaciones Central implementa un estricto mecanismo de Timeout en cada etapa del cálculo. El proceso se inicia al enviar la instrucción específica al servidor asignado según el flujo de la ecuación cuadrática; en ese instante, el orquestador inicia un cronómetro de espera activa con un tiempo límite T . Si el socket no recibe una respuesta válida antes de que expire dicho tiempo, el servidor central marca inmediatamente al nodo como *No Disponible* en su estado interno y activa el protocolo de sustitución. Acto seguido, la operación fallida es reenviada automáticamente al servidor sustituto designado asegurando que el cliente reciba un resultado sin interrupciones, siempre que exista al menos un nodo operativo en la red. Por lo que considerando las posibles combinaciones entre el servidor de operaciones 1, 2 y 3, se consideran los siguientes roles:

OPERACIÓN ORIGINAL	SERVIDOR PRINCIPAL	SUSTITUTO 1	SUSTITUTO 2
RAÍZ ($\Delta = \sqrt{B^2 - 4AC}$)	Op1	Op2	Op3
SUMA/RESTA NUMERADOR	Op2	Op3	Op1
DIVISIÓN FINAL	Op3	Op1	Op2

A continuación, se detallan las rutas que toma la información cuando el servidor central detecta un fallo mediante el mecanismo de timeout, considerando lo anterior:

- Escenario con 0 fallos:

Op1	Op2	Op3	Flujo
✓	✓	✓	Flujo normal: Op1 → Op2 → Op3

El proceso inicia cuando el cliente envía al coordinador los coeficientes a, b y c de la ecuación cuadrática. El coordinador actúa como un orquestador y divide el cálculo en varias etapas, delegando cada una a un servidor de operación distinto.

Primero, el coordinador envía una solicitud al servidor op1 para calcular la raíz de la discriminante. Una vez que op1 devuelve el resultado (`sqrt_d`), el coordinador continúa con la segunda etapa y solicita al servidor op2 el cálculo de los numeradores de la fórmula cuadrática. Tras recibir estos valores, el coordinador envía la última operación al servidor op3, que realiza la división final para obtener las raíces x_1 y x_2 .

- Escenario con 1 fallo:

Op1	Op2	Op3	Flujo alterno
✗	✓	✓	Raíz: Op2 → Suma/Resta: Op2 → División: Op3
✓	✗	✓	Raíz: Op1 → Suma/Resta: Op3 → División: Op3
✓	✓	✗	Raíz: Op1 → Suma/Resta: Op2 → División: Op1

Este diagrama ilustra el comportamiento del sistema cuando uno de los servidores de operación falla. En el ejemplo, el coordinador intenta ejecutar una operación (por ejemplo, el cálculo del discriminante) en un servidor principal. Sin embargo, dicho servidor no responde o se produce un timeout. El coordinador detecta este fallo y marca al nodo como no disponible.

A continuación, el coordinador reintenta la misma operación con un servidor alternativo, siguiendo el plan de sustitución definido. Cuando el segundo servidor responde correctamente, el proceso continúa con normalidad.

- Escenario con 2 fallos:

Op1	Op2	Op3	Flujo alterno
✗	✗	✓	Op3 hace TODO
✗	✓	✗	Op2 hace TODO
✓	✗	✗	Op1 hace TODO

En esta situación, el coordinador detecta que ya no es posible continuar con el cálculo distribuido por etapas. Para evitar una falla total del sistema, activa un modo de contingencia en el cual el único servidor operativo ejecuta el cálculo completo de la ecuación cuadrática en una sola operación.

El coordinador envía una solicitud especial de tipo `full_quadratic` al nodo restante, el cual realiza todos los pasos matemáticos internamente y devuelve directamente las raíces x_1 y x_2 . El coordinador, a su vez, envía el resultado final al cliente.

- Escenario con 3 fallos:

Op1	Op2	Op3	Acción
✗	✗	✗	Respuesta al cliente: "Perdona la demora, intenta más tarde"

Cuando se presentan tres fallos simultáneos en los servidores de operación, el sistema entra en un escenario de indisponibilidad total, ya que no existe ningún nodo capaz de ejecutar las operaciones requeridas. En esta situación, el coordinador intenta primero aplicar el mecanismo de tolerancia a fallos, pero al detectar que todos los servidores han sido marcados como caídos, concluye que no hay recursos disponibles para continuar con el cálculo distribuido ni para activar el modo de contingencia con un solo nodo. Como resultado, no se puede completar ninguna etapa del proceso matemático.

Ante este escenario, el coordinador finaliza la solicitud devolviendo al cliente un mensaje de error claramente identificado, indicando que no existen nodos disponibles para atender la operación. Este comportamiento evita bloqueos indefinidos o esperas innecesarias y garantiza que el cliente reciba una respuesta coherente con el estado real del sistema. De esta manera, el sistema mantiene consistencia y transparencia, informando explícitamente la imposibilidad de servicio cuando se supera el límite de tolerancia a fallos para el cual fue diseñado.